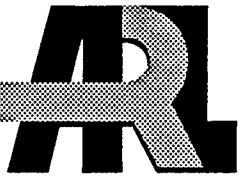
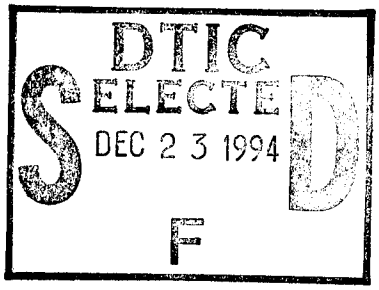
ARMY RESEARCH LABORATORY



Parallel Computing on Various Platforms - A Case Study

B. N. Srivastava

ARL-TR-633 November 1994



APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

19941219 005

NOTICES

Destroy this report when it is no longer needed. DO NOT return it to the originator.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The use of trade names or manufacturers' names in this report does not constitute endorsement of any commercial product.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 1994	Interim, Feb–Oct 94

4. TITLE AND SUBTITLE
Parallel Computing on Various Platforms - A Case Study

5. FUNDING NUMBERS
DAAA15-91-C-0082

6. AUTHOR(S)
B. N. Srivastava

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-CI-CA
Aberdeen Proving Ground, MD 21005-5067

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-OP-AP-L
Aberdeen Proving Ground, MD 21005-5066

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
ARL-TR-633

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Parallel computing offers an attractive means for high-performance computing to alleviate the current bottleneck in speedup potential of large application codes for practical problem solving. The challenge of actual implementation, however, can be significant. The objective of this report is to address the issue of porting and parallelizing a Computational Fluid Dynamics (CFD) application code on several supercomputing platforms such as Cray C-90, Cray Y-MP, KSR1, CM-5 and Intel PARAGON. This overall objective is achieved through first emulating the application code by concocting a model test code, in an effort to understand the influence of data structure on the code performance. The conclusions derived from this are then utilized to restructure the 2-D/3-D CFD code such that the final version is transportable to a wider class of platforms.

14. SUBJECT TERMS	15. NUMBER OF PAGES
high-performance computing, parallel computing, computational fluid dynamics, turbomachinery, computers, problem solving	42
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

INTENTIONALLY LEFT BLANK.

TABLE OF CONTENTS

LIST OF FIGURES

Accesion For
NTIS CRA&I
DTIC TAB
Unannounced
Justification
By
Distribution /
Availability Codes
Dist Special
A-1

INTENTIONALLY LEFT BLANK.

1. INTRODUCTION

Parallel computing offers an attractive means for high-performance computing to alleviate the current bottleneck in speedup potential of large application codes for practical problem solving. The challenge of actual implementation, however, can be significant and can depend on the nature of application. Computational Fluid Dynamics (CFD) is one area which can effectively use the inherent speedup potential for solving complex fluid dynamic problems of practical interest. The effort of porting and parallelizing such codes, however, depends on the specific application and the extent of scalable aspects of the code. The general rules of scalable programs that are known today were nonexistent a few years ago. This implies that older codes may require more significant modifications than new ones (that are written in array-style FORTRAN). The real issue of such porting is to first determine the level of modification within the constraints of the application schedules. In most cases, significant effort may be required to realize the full potential of a chosen platform. A priori, such modifications are not obvious, and one is typically stuck with "learn as you proceed" syndrome. On most platforms, translation softwares are available, but extreme caution needs to be exercised since these softwares can handle only straightforward array-based programming styles.

A reasonable strategy for application engineers is to emulate their code by concocting a model test code that can be exercised on various platforms. This approach serves two purposes:

(1) A user can exercise this code in an interactive environment to quickly learn various options for a given platform.

(2) By making minor modifications in the array structure of the code, the user can develop an understanding of the influence of data structure on the code performance. This allows one to chose the desirable data structure for the specific platform.

The philosophy of testing model codes (as opposed to large application codes) not only offers insight into planning the required modifications for large application codes but also allows the user to quickly establish the relationships between various architectures and code data structures. This paper describes this approach in relation to a large 3-D/2-D CFD application code for a variety of supercomputing platforms such as the Cray C-90, Y-MP, KSR1, CM-5, and Intel PARAGON. Based on these studies, specific conclusions and recommendations that are useful for the parallel computing community are made.

2. CFD CODE DESCRIPTION

It is relevant to highlight the CFD code that will be used during this technical effort. Explicit TVD symmetric differencing schemes with the Baldwin-Lomax turbulence model are used to solve the Reynolds averaged Navier-Stokes (NS) equations for steady and unsteady flows. Further details of numerical formulation, boundary approaches, and applications for this code can be found in Srivastava and Bozzola (1984, 1987); Srivastava (1987); Srivastava, Maia, and Moran (1988); and Srivastava et al. (1992). For this effort, the 2-D/3-D versions of the code for turbomachinery applications have been used. A version of the cascade code was used to first run a practical case of interest. The test case involved NASA Energy Efficient Engine (EEE) rotor cascades that have been extensively tested in cascade tunnels (Kopper et al. 1981). Some typical results, comparisons with experiments, and grids used for this computation are shown in Figures 1–4 for demonstration purposes. The purpose here is to highlight the specific application for the CFD code and its relationship to the turbomachinery design environment. Speedup of such CFD codes is of significant interest in order to reduce the overall cost of design iterations. Parallel computing offers an attractive alternative to achieve this goal.

3. CFD CODE EMULATION MODEL

The highlights of the time-asymptotic NS code for turbomachinery applications, the 2-D/3-D version, can be emulated by observing the following guidelines:

(1) Grid setup and the attendant storage - compute and write step.

(2) Flux setup using previous time step - compute and write step.

(3) State vector inversion step for new time step - recall and compute step.

The current NS code is a unique combination of 2-D/3-D array styles and linear arrays that were designed to:

(1) run on traditional shared memory vector computers which had storage limitations for large problems. Minimization of memory as well as enhancement of vector capability were the major objectives.

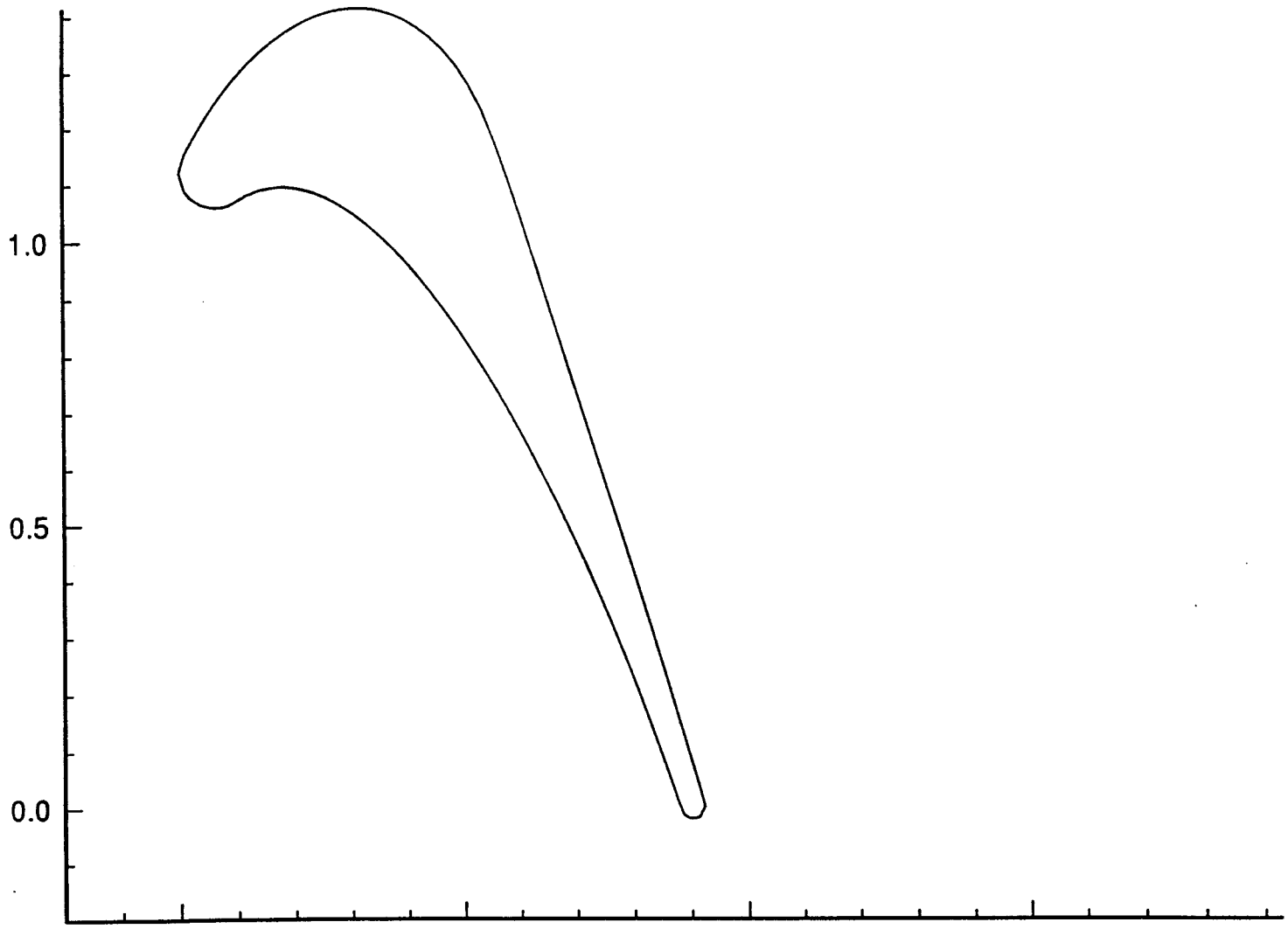


Figure 1. Blade geometry for EEE cascade.

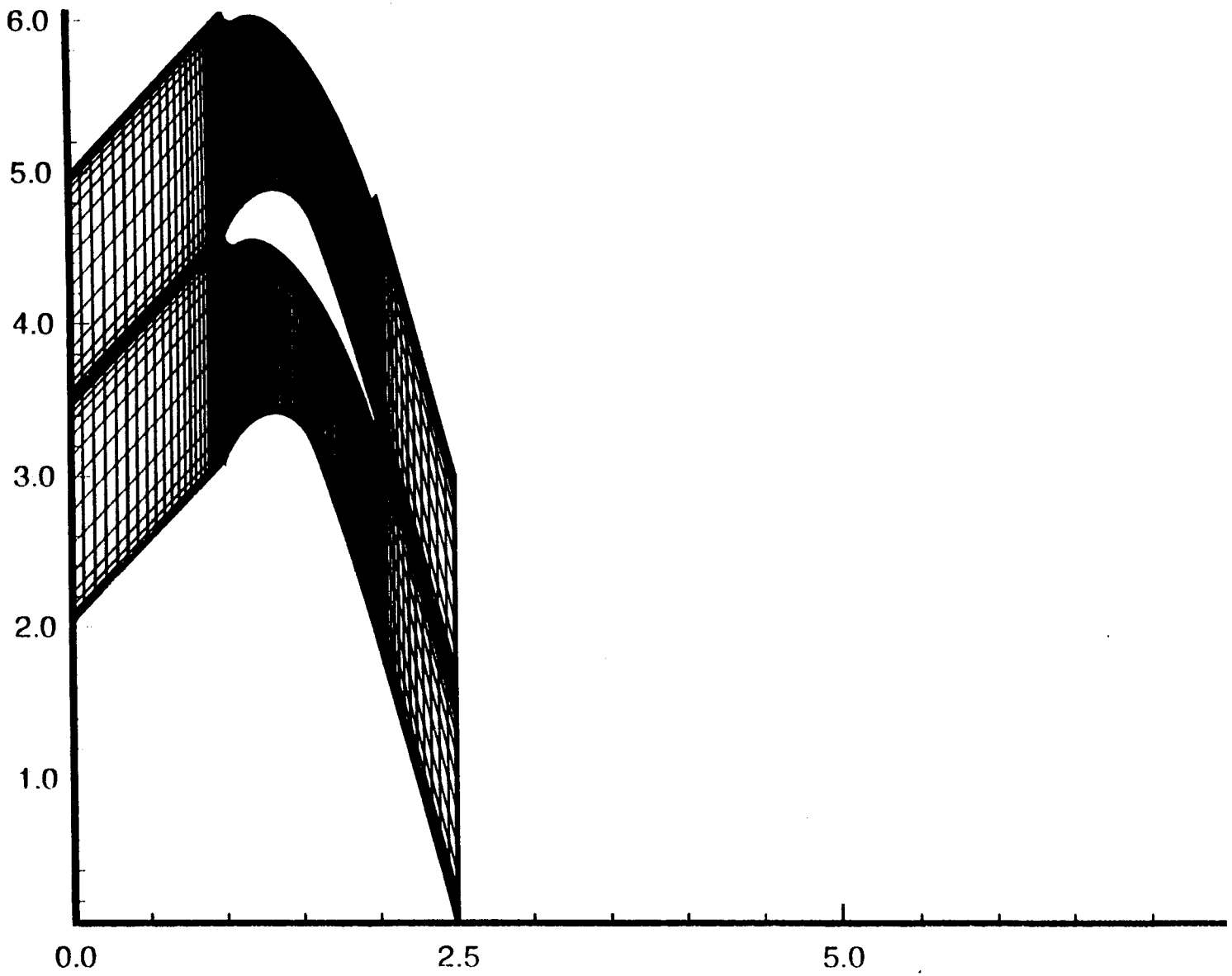


Figure 2. H-grid topology for rotor cascade geometry.

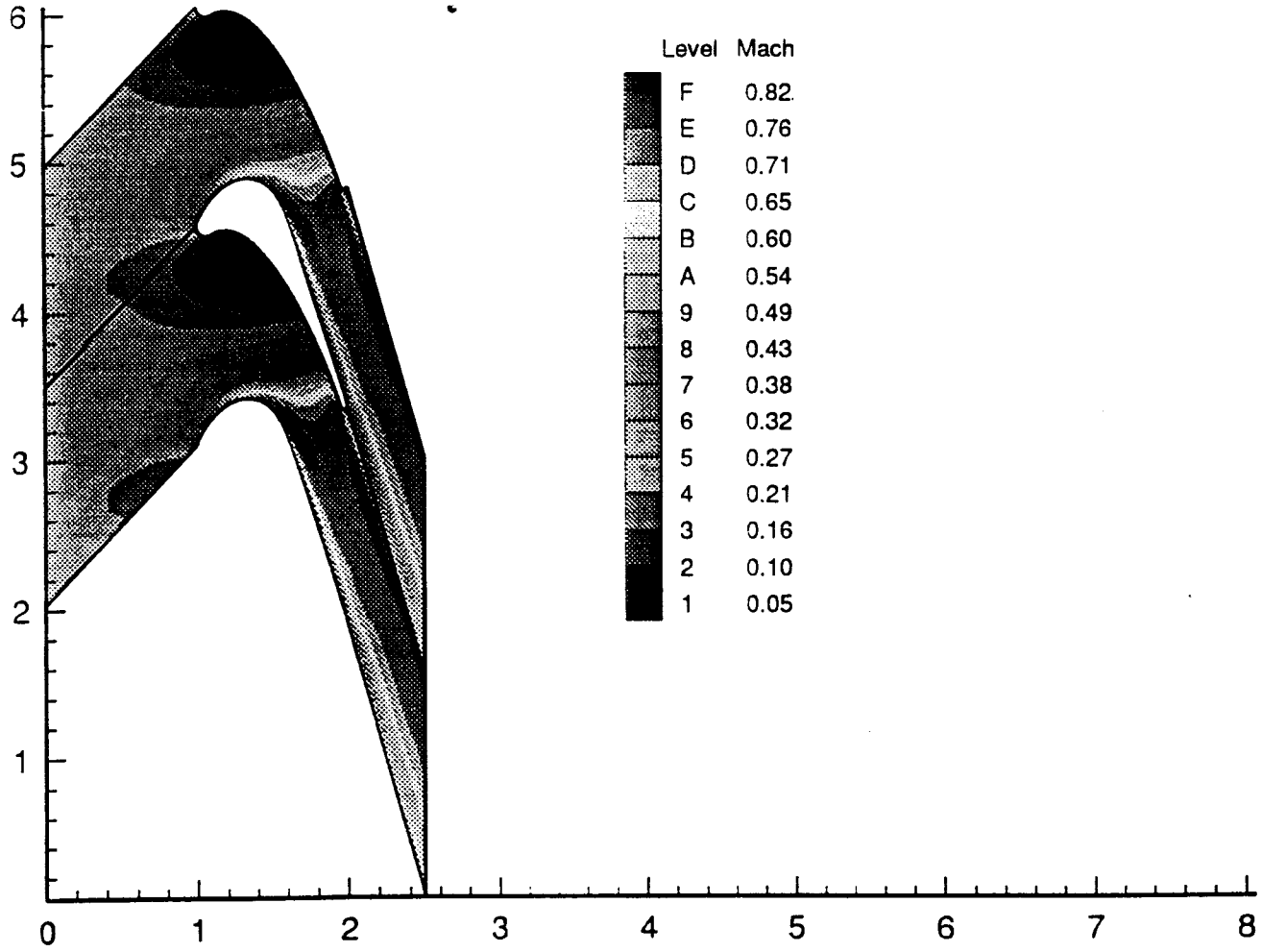


Figure 3a. Mach number contour levels for rotor cascade.

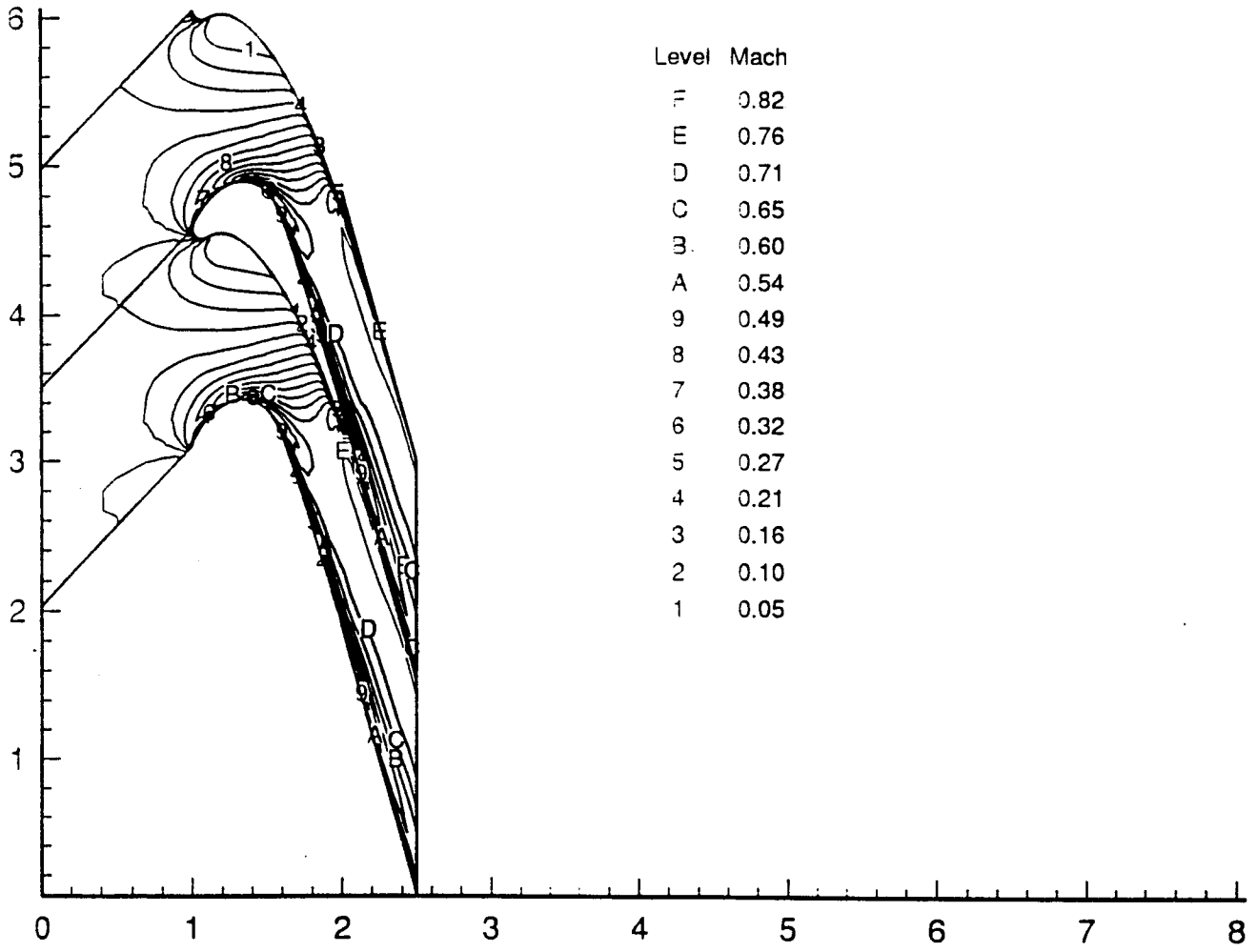


Figure 3b. Mach number contour lines for rotor cascade.

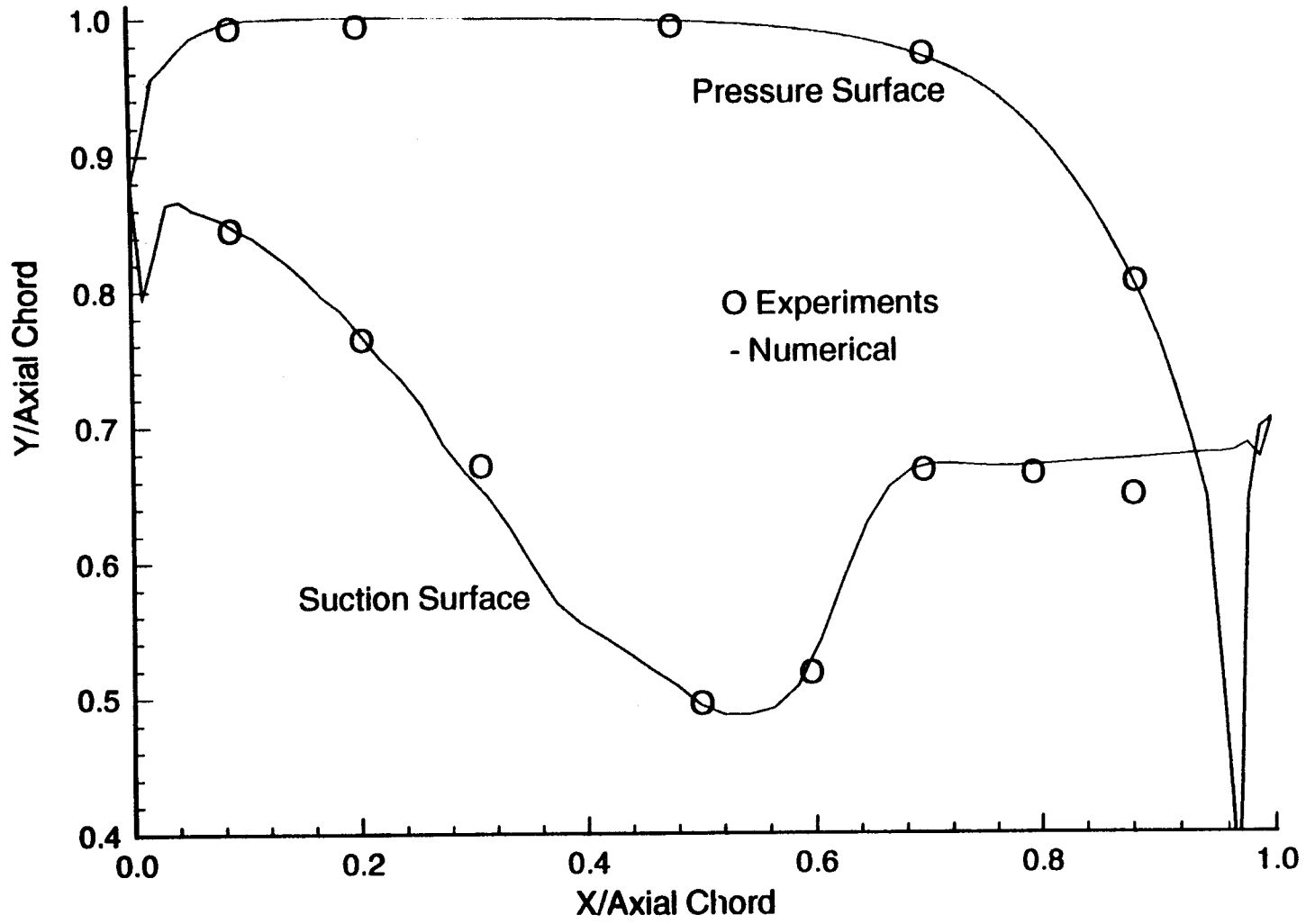


Figure 4. Comparison of cascade pressure predictions with experiments.

(2) yield optimal performance on a few platforms such as MASSCOMP, ALLIANT, STARDENT, IBM, and Cray-1.

(3) extensively uses local variables to avoid repeat calculations or repeat memory reference.

(5) exploit parallelism that was limited to shared-memory-type machines.

4. TESTING PROCEDURES

There are certain ground rules that must be established to do testing of the emulated code on various platforms:

(1) The code is being tested on each platform with a view to establish a probable efficient data structure layout that gives optimum performance on a given platform.

(2) The code will be run in a shared environment with no special request for this testing.

(3) No timing comparisons of various platforms will be made since full potential of each machine can not be realized in a short study like this.

(4) The basic code will be ported to each machine and run in a serial and parallel manner. The experiences and results will be documented.

(5) For parallel computing, the serial code will be modified to provide correct results on multiple nodes. Extensive modifications at advanced level is not considered at this point.

5. CRAY Y-MP AND C-90 STUDIES

Full potential of the 2-D and 3-D codes can be demonstrated through a moderate effort on Cray systems. The greatest advantage of Cray systems is the user support software for vector and parallel computing. The emulated model test code was not exercised on these systems, but rather the full code was optimized. The following steps outline the macro, micro, and auto-tasking procedures to optimize the code:

(1) Compiler optimizations with -Zp switch were totally insufficient to get any speedup of either 2-D or 3-D codes. The reasons were the linear arrays which prevented compiler analysis of data dependencies. The speedup was only 20% of the original code.

(2) Each individual routine was then analyzed using FPP analyzer. This pointed out the need for specific CFPP$ commands that were required to obtain vectorizing and parallelizing of each loop in the subroutine. For 3-D loops, parallel outer loops operated on the largest index.

(3) Small inner loops were inlined via CFPP$ INLINE commands to allow longer vector dimensions. The inner loop, however, is not the longest in the codes, and a switch in index is not straightforward due to linear arrays. This can be done later for larger potential gains.

(4) Subroutine calls from a parent were parallelized via CFPP$ CNCALL directives that ensured ignoring of compiler data dependency checks.

(5) Several smaller turbulence model routines had to be inlined manually to alleviate FPP analyzer errors and its refusal to vectorize/parallelize some loops.

(6) The process was complete when each loop in each routine was either vectorizable and parallelizable as shown in FPP generated files.

(7) Figures 5 and 6 show the results of such optimizations yielding a factor of nearly 3 to 4 over the original code. The entire effort was completed in two weeks for both codes, and the potential for even higher factors still exists.

5.1 2-D/3-D Code Structure Studies. The array storage styles in computer codes can affect computing performance. The platform architecture plays a crucial role in this regard. Data parallel styles are now much more prevalent than linear arrays. Many compilers are now specifically designed to recognize the data parallel styles. The current code was written a few years ago in mostly linear array style through optimizations. The complexity in linear array styles is numerous due to nonapparent data utilization. For this effort, it was found necessary to address this issue up front to make a judicious choice of continuing this style further. For this reason, an up-front effort was undertaken to change the entire code from one style to another. The rationale for this study is discussed next.

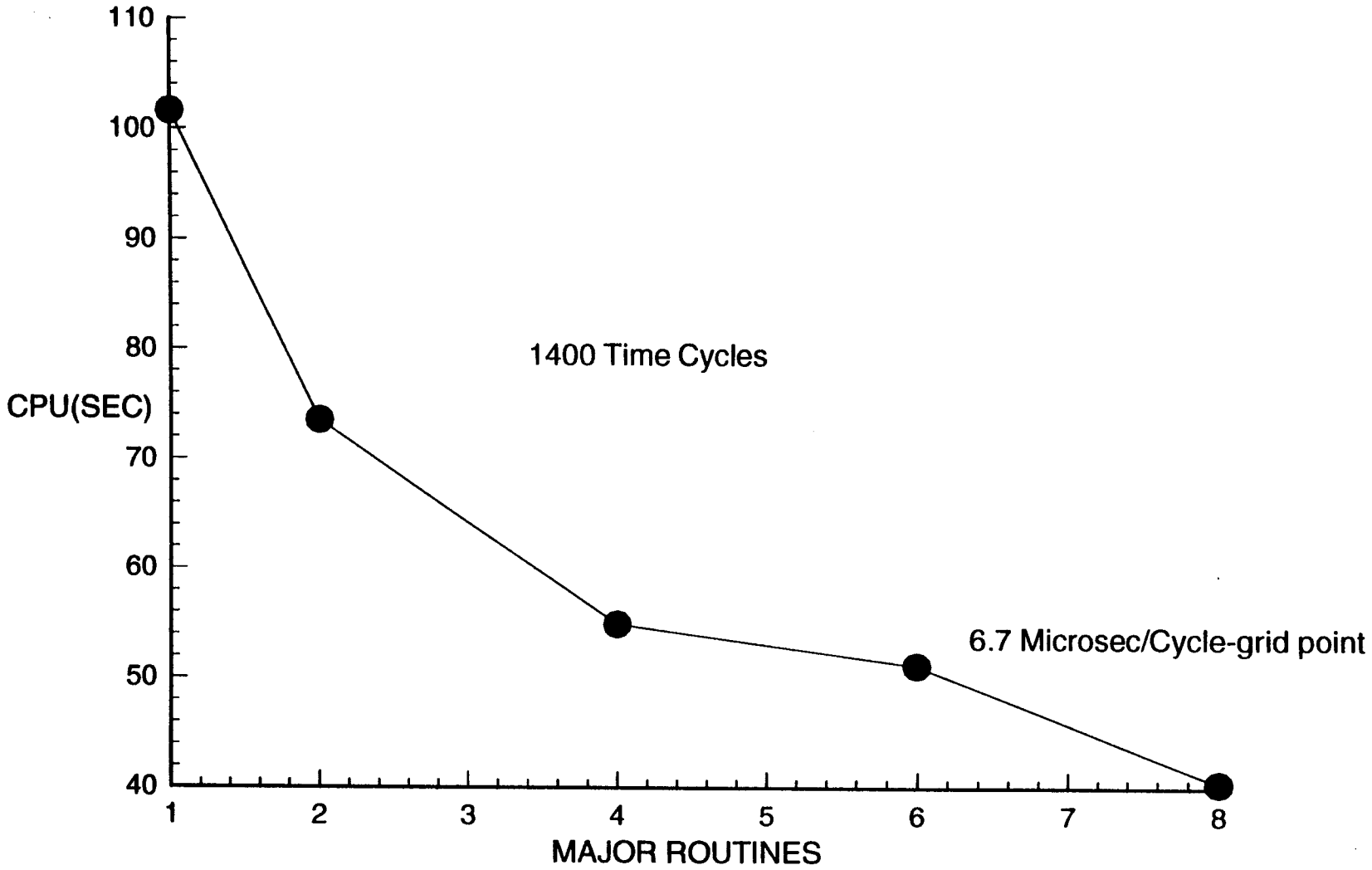


Figure 5. 2-D cascade code vector and parallel study for Cray C-90.

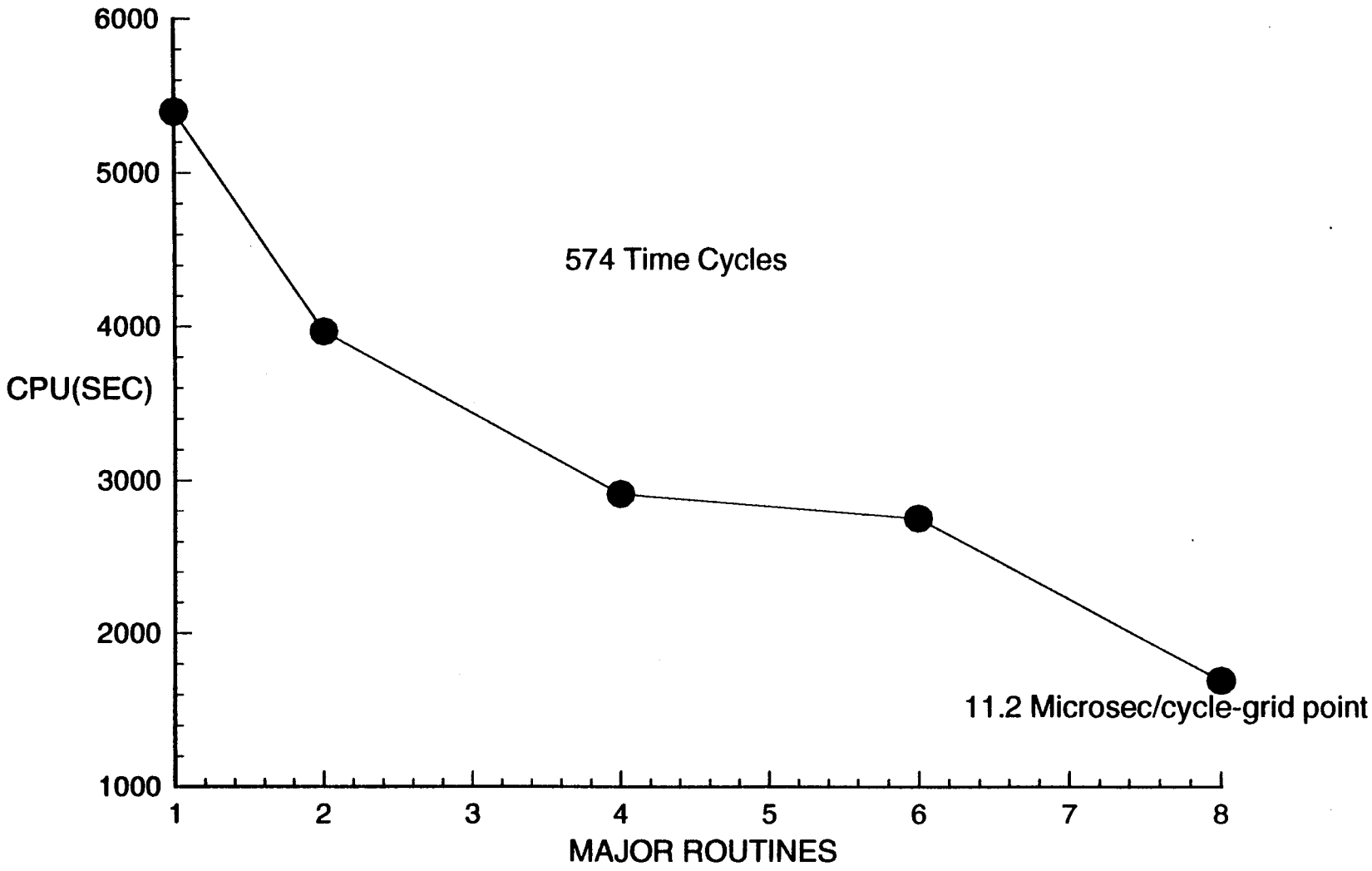


Figure 6. 3-D cascade code vector and parallel study for Cray C-90.

Traditional wisdom of avoiding repeat arithmetic operations are typically handled through data storage. For most computing platforms, however, there is a tradeoff between memory and CPU requirements. Marginal increase in CPU for substantial savings in memory could be a desirable goal for high-performance computing. This becomes more so for parallel computing where the compute-to-communicate ratio needs to maximized. For an explicit CFD code, there are numerous ways to achieve an optimized code structure which offers a reasonable balance between the memory and CPU requirements. This section describes a study that provides a basis for code structure selection for an explicit algorithm.

Vector and parallel constructs in a 3-D explicit code are achieved through:

(1) flux computation using previous time data. This construct is completely parallel. Computed fluxes can be stored as HFLX(I, J, K) for the next step.

(2) inversion using fluxes. Storing fluxes followed by inversion is serial in nature. For a 3- to 5-point explicit algorithm, flux computation and inversion can proceed in parallel for every chunk of 3–5 points. Overlapping points can be repeat computations to avoid data fetch or wait process. Naturally, there are tradeoffs. These tradeoffs are described later.

Other code structure issues relate to linear vs. 3-D/2-D arrays. This code structure study correlates very well with the model code structure studies that have been discussed later for various other computing platforms such as KSR1, PARAGON, and CM-5.

For code structure studies, a Cray Y-MP was used for rapid assessments. Figure 7 shows the run time results for various code structures using vectorization as well as multitasking. Figure 8 shows the run time results with vectorization only (without invoking the multitasking options). Note from this figure that there is an increase in run time for Case II when only vectorization is used for the code. This is caused by the repeat overlapping computations in Case II (as described later) that are not compensated due to lack of multitasking. The various versions of the code are described as follows:

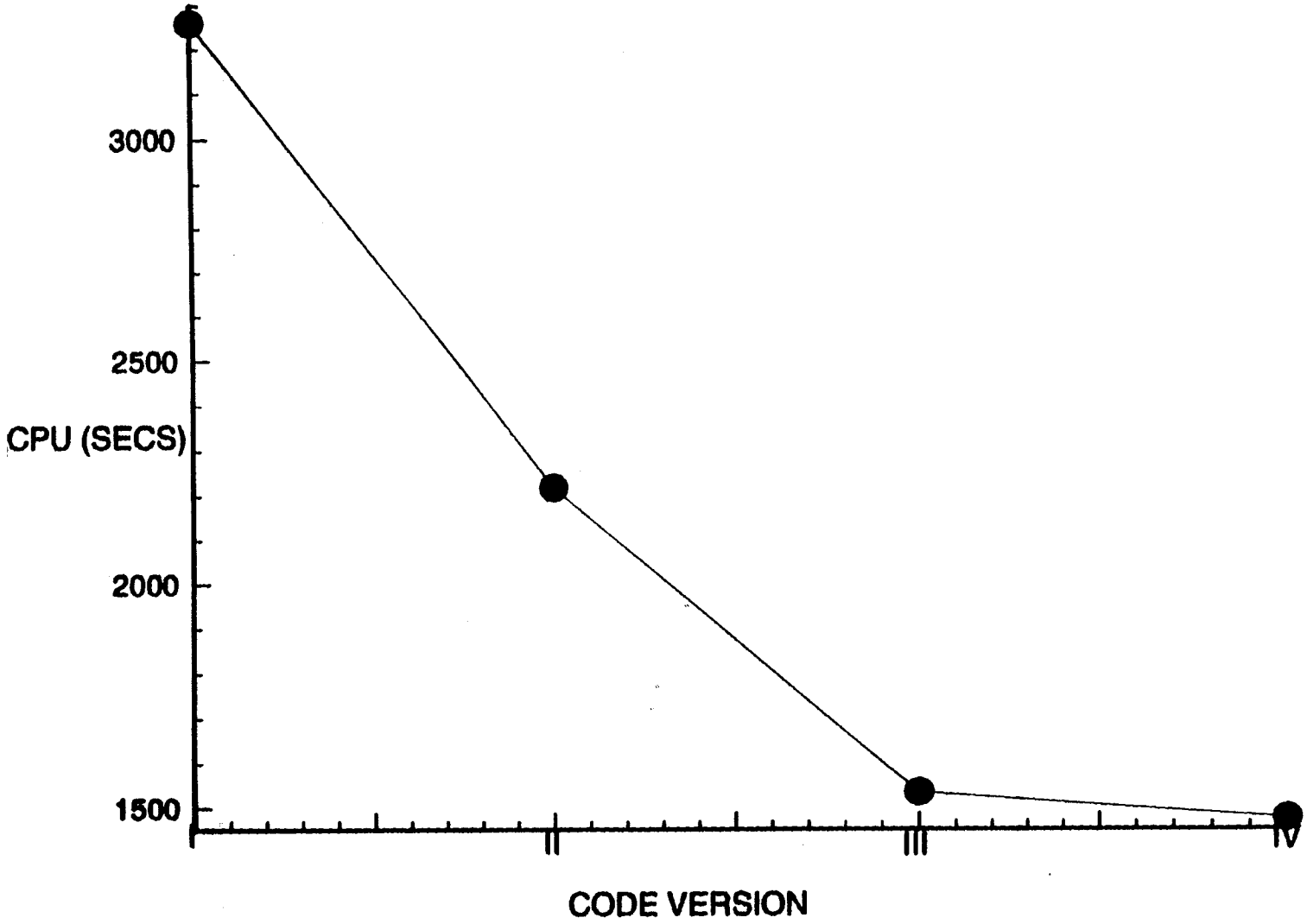


Figure 7. Effect of code structure on performance using vectorization and multitasking for Cray Y-MP.

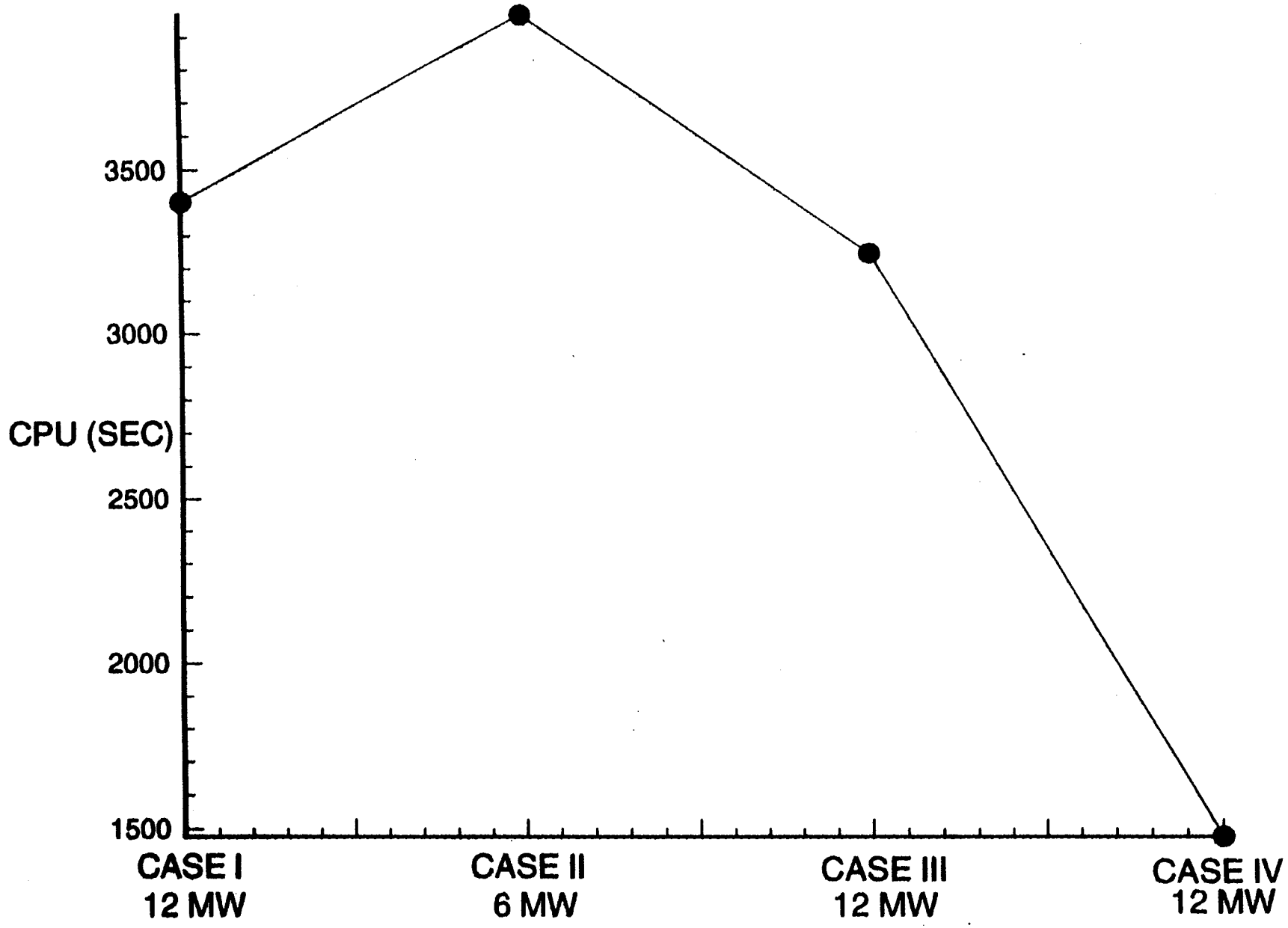


Figure 8. Effect of 3-D code structure on performance using vectorization for Cray Y-MP.

(1) **CASE I**

Contains all serial data structures. Fluxes are stored as HFLX(imax*jmax, kmax). These are computed first for all grid nodes followed by inversion. The two steps are serial in nature for the entire domain. For 101 × 51 × 51 grid points, the memory required is 12 Mwords.

(2) **CASE II**

For 3-point formulation, fluxes are computed for three z-planes and stored as HFLX(imax*jmax, 3). Parallel flux computation and inversion is achieved for every three points. Repeat flux computations are performed to avoid synchronization of inversion process for overlapping points. Memory requirements for this version is six Mwords.

(3) **CASE III**

For this version, flux computation and inversion is similar to Case I, but state vectors were all transformed from linear arrays to 3-D arrays as in V(I, J, K); there are 10 of these. For Cases I and II, they were stored as V(10, I*J, K). Memory requirements for this version is six Mwords.

(4) **CASE IV**

For this version, all linear arrays were transformed to 3-D arrays. No linear arrays were present here. Fluxes are computed as HFLX(I, J, K), and inversion follows serially. Memory requirements for this version is six Mwords.

5.2 Conclusions. The studies conducted here suggest that:

(1) Vectorization and parallel effort gave a substantial gain in computing performance, on the order of four.

(2) Changes in code structure then realized another factor of 2 enhancement in computing performance. This implies that the data parallel structures V(I, J, K) are preferable for these types of computing platforms.

The objective of this effort was to demonstrate the potential of speedup for this code. It also demonstrates the influence of data structures on computing performance for parallel processing. The next natural question relates to exportability of such conclusions for other types of platforms, specifically massively parallel systems. This provides a natural entry point for the next step.

6. PRELUDE TO MASSIVELY PARALLEL EFFORT

The rapid success on Cray systems was a motivating factor to first go to the KSR1 system because of its similar architecture for parallel computing using the 2-D/3-D codes. The experience, however, was not pleasant since KAP, an equivalent to possibly FPP, failed miserably on the code. In fact, KAP generated codes that were wrong in some instances and, in most instances, it created codes with ORDER command in tiling, instructing the compiler to essentially execute in serial. Ultimately, the KAP effort was abandoned in favor of manual tiling. Manual tiling yielded results for 2-D code that were beginning to show some favorable results. Figure 9 shows the result for 2-D code. Note that the scaling begins to saturate with about eight processors. To test whether compute-to-communicate ratio is low for this code, a new code with large dummy work load was created and tested. This result is shown in Figure 10. Note that the trend remains the same. Nonetheless, 3-D code effort was initiated to understand KSR1 behavior in comparison with Cray. Figure 11 shows the run time results obtained on one processor of KSR1 using various versions of the code as described before. This figure shows a trend opposite to what was observed for Cray (see Figure 7). Apart from this, on multiprocessors, no significant speedup has been observed for KSR1. Obviously, a lot more questions need to be answered to unravel the observed behavior on KSR1. The questions are:

(1) What are the most desirable code structures common to all parallel platforms?

(2) Is there any such commonality among the various platforms?

(3) How can the current code be restructured to run effectively (if not optimally) on all these platforms?

This can only be answered quickly by using a model test code that emulates the CFD code. This is done next.

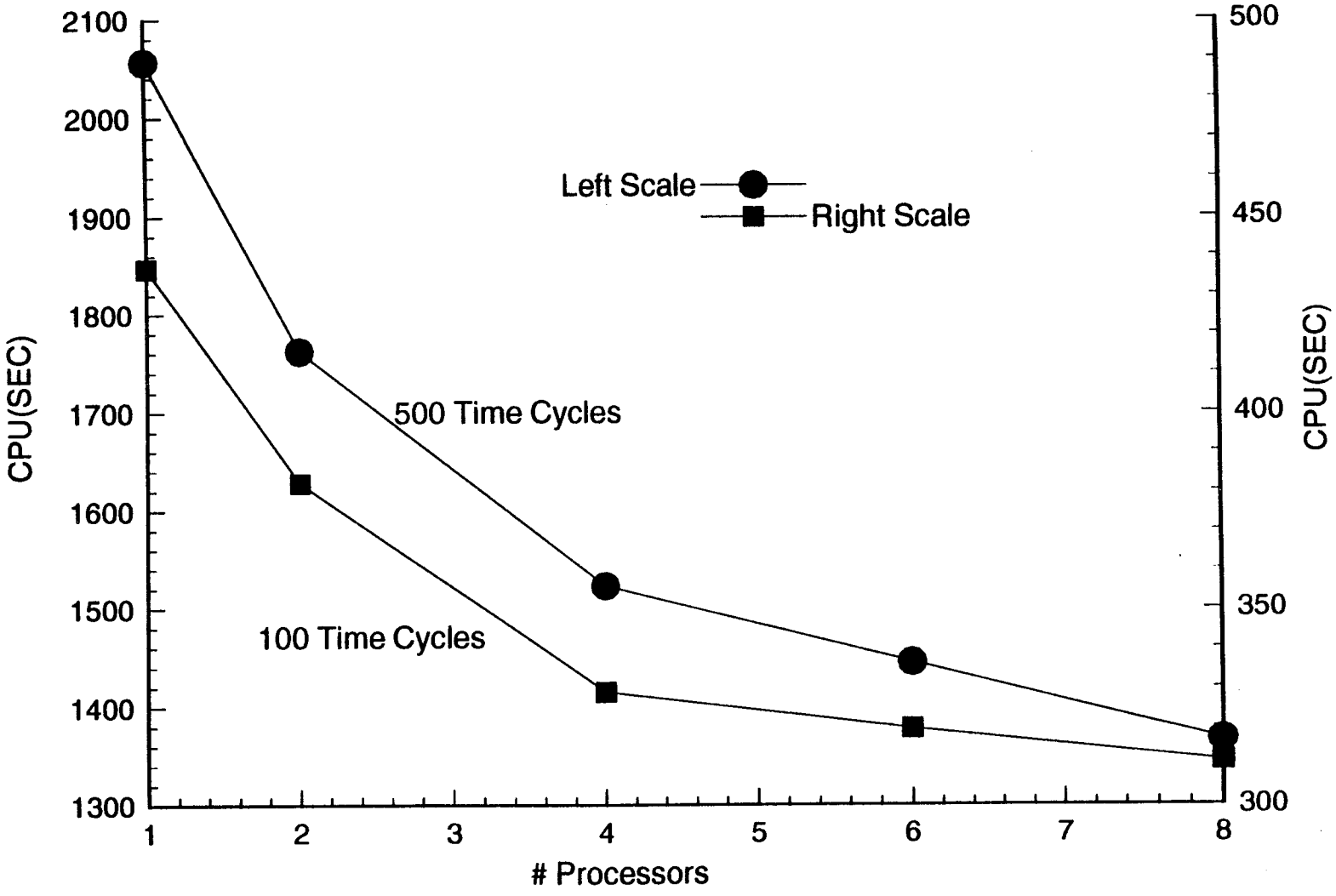


Figure 9. Parallel studies for 2-D cascade code using KSR1.

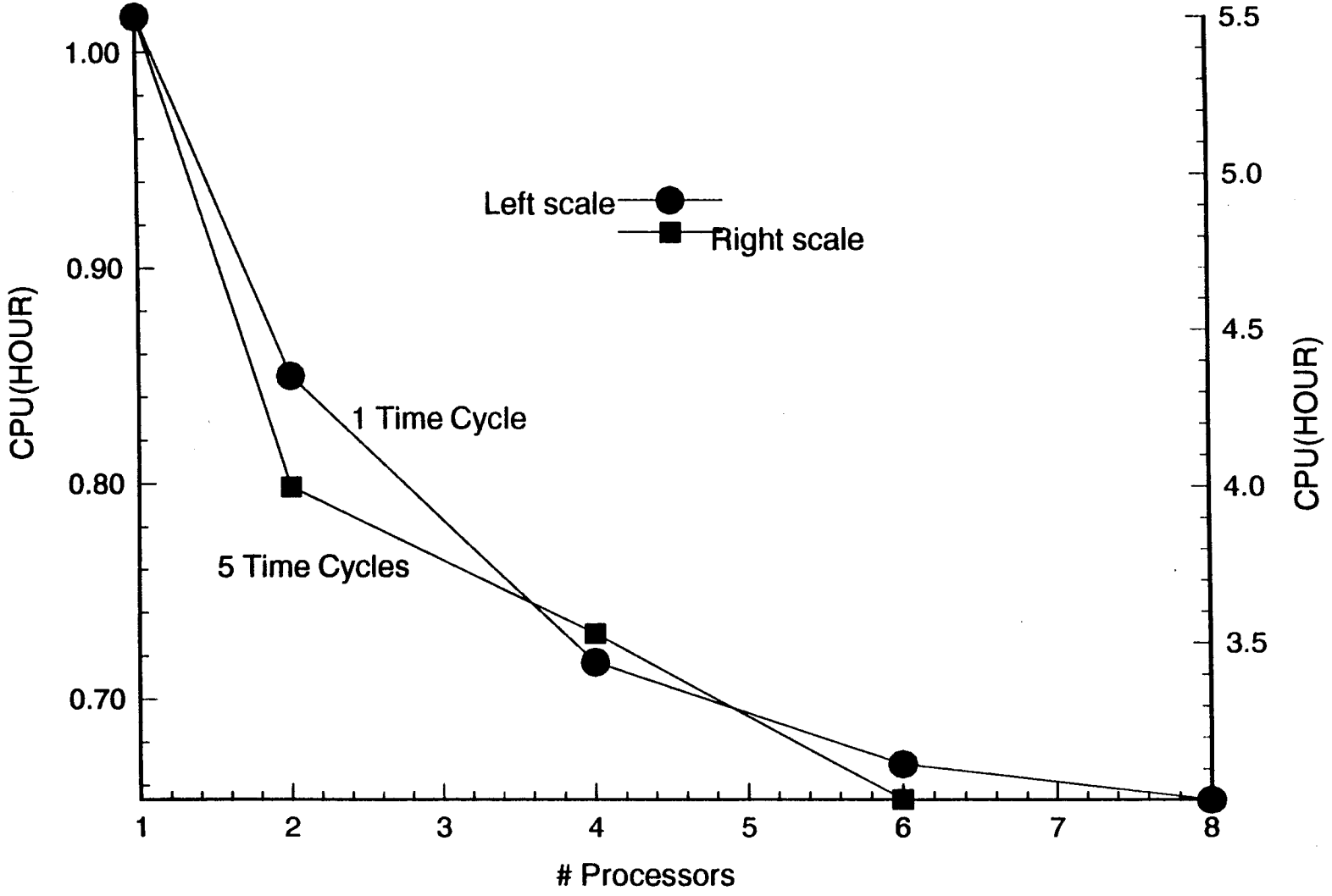


Figure 10. Parallel studies for 2-D cascade code with dummy work loads using KSR1.

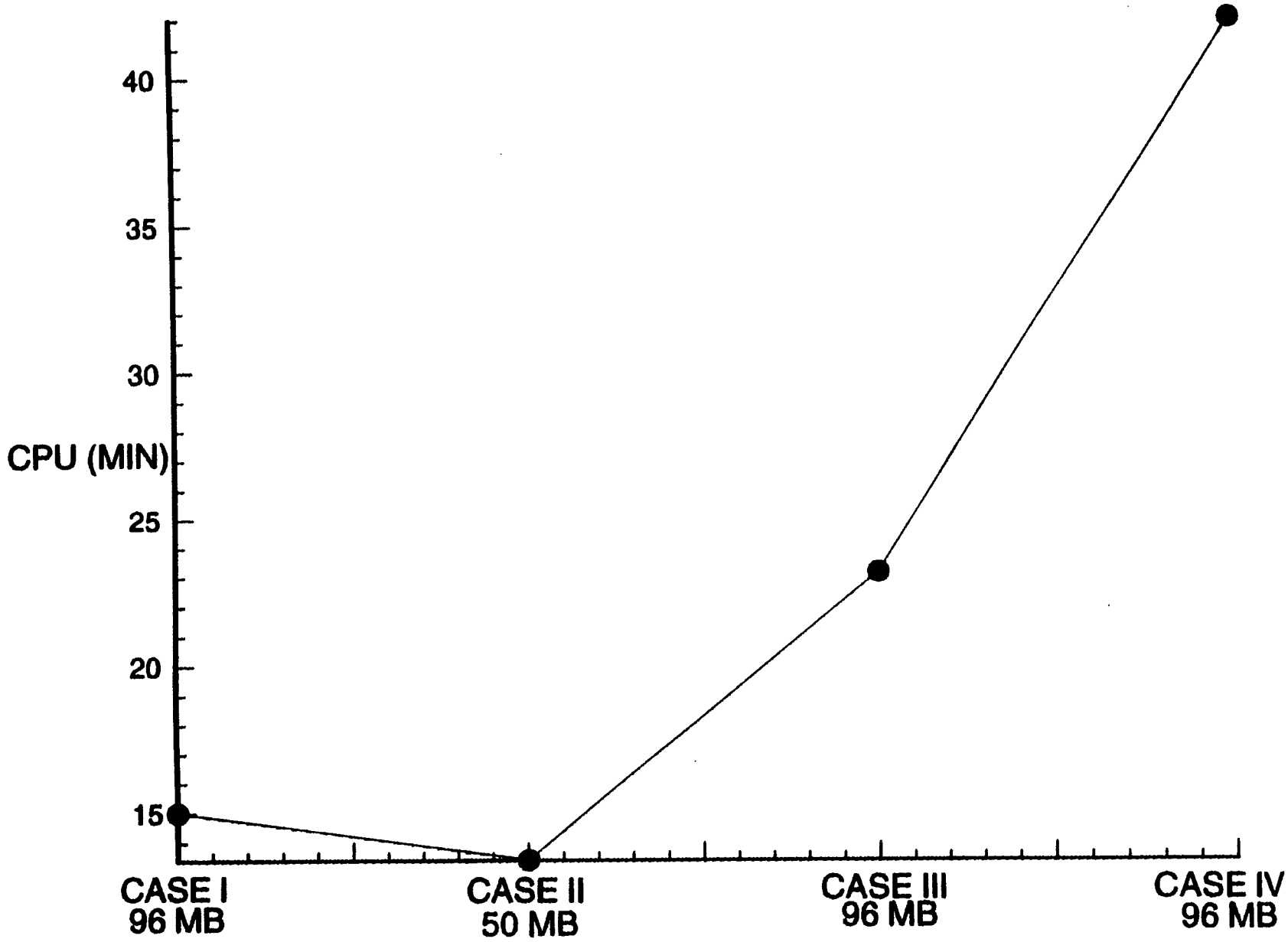


Figure 11. Effect of 3-D code structure on performance for 1-node KSR1.

7. KSR1 PERFORMANCE STUDY

For KSR, both tiling strategy and parallel region approach were tested. No specific advantage of one over the other was observed. For all test cases:

jmax=5000, kmax=500

time normalized with 1 processor (proc)

7.1 Case I.

Array Structure - y(3, kmax*jmax)

Code:

```
do j
do k
y(1,k*j)=
y(2,k*j)=
y(3,k*j)=
.....
end do
end do
```

Results: 1 Proc Time = 3.6 Minutes
Memory Needed 62 Mb

Procs	Time	Comment
1	1.00	Pbss memory
2	0.71	three times
5	0.65	larger.
10	0.94	

7.2 Case II.

Array Structure - y1(j,k), y2(j,k), y3(j,k), Mismatched loop index

Code:

```
do j
do k
y1(j,k)=
y2(j,k)=
y3(j,k)=
.....
end do
end do
```

Results: 1 Proc Time = 15.6 Minutes
Memory Needed 22 Mb

Procs	Time	Comment
1	1.00	1 Proc time
2	0.87	very high.
5	0.54	
10	0.34	

7.3 Case III.

Array Structure - y1(k,j), y2(k,j), y3(k,j), Matched loop index

Code:

```
do j
do k
y1(k,j)=
y2(k,j)=
y3(k,j)=
.....
end do
end do
```

Results: 1 Proc Time = 4.3 Minutes
Memory Needed 22 Mb

Procs	Time	Comment
1	1.00	Inner loop
2	0.53	aligned with
5	0.30	lead index k.
10	0.20	

7.4 Case IV.

Array Structure - y1(k*j), y2(k*j), y3(k*j), Matched loop index

Code:

```
do j
do k
y1(k*j)=
y2(k*j)=
y3(k*j)=
.....
end do
end do
```

Results: 1 Proc Time = 2.4 Minutes
Memory Needed 22 Mb

Procs	Time	Comment
1	1.00	Best result
2	0.51	with linear
5	0.24	arrays.
10	0.15	

7.5 Case V.

Array Structure - y1(k*j), y2(k*j), y3(k*j), Mismatched loop index

Code:

```
do k
do j
y1(k*j)=
y2(k*j)=
y3(k*j)=
.....
end do
end do
```

Results: 1 Proc Time = 9.3 Minutes
Memory Needed 22 Mb

Procs	Time	Comment
1	1.00	1 Proc time
2	0.17	very large.
5	0.08	
10	0.05	

7.6 Case VI.

Array Structure - y(3* kmax*jmax)

Code:

```
do j
do k
y(1*k*j)=
y(2*k*j)=
y(3*k*j)=
.....
end do
end do
```

Results: 1 Proc Time = 3.3 Minutes
Memory Needed 62 Mb

Procs	Time	Comment
1	1.00	Pbss memory three
2	0.68	times large, 10 procs
5	0.39	results wrong.
10	0.36	

7.7 Conclusions. The overall results from above are shown pictorially in Figure 12. For KSR1 applications, the best performance is obtained with linear arrays with the lead index aligned with inner loop index. This relates to subpaging of the data in the loop which are used effectively once they are brought in to subpage. The reason why linear array is better than 2-D array is not clear. Also, when all variables are rolled into one array, the performance results are not only bad but are also wrong for a large number of processors. Since in a large code, the linear array size can be very long, a safe recommendation would be to go for 2-D/3-D arrays.

8. PARAGON PERFORMANCE STUDY

The Intel PARAGON at Wright-Patterson Air Force Base was used for this performance study. It is relevant to point out the following:

(1) Domain decomposition is an important aspect of parallel computing for distributed memory environment; and hence, memory available on each processor is an important consideration before embarking on programming. On PARAGON, each processor can accommodate 32 Mb, but with system requirements no more than 30 Mb should be used.

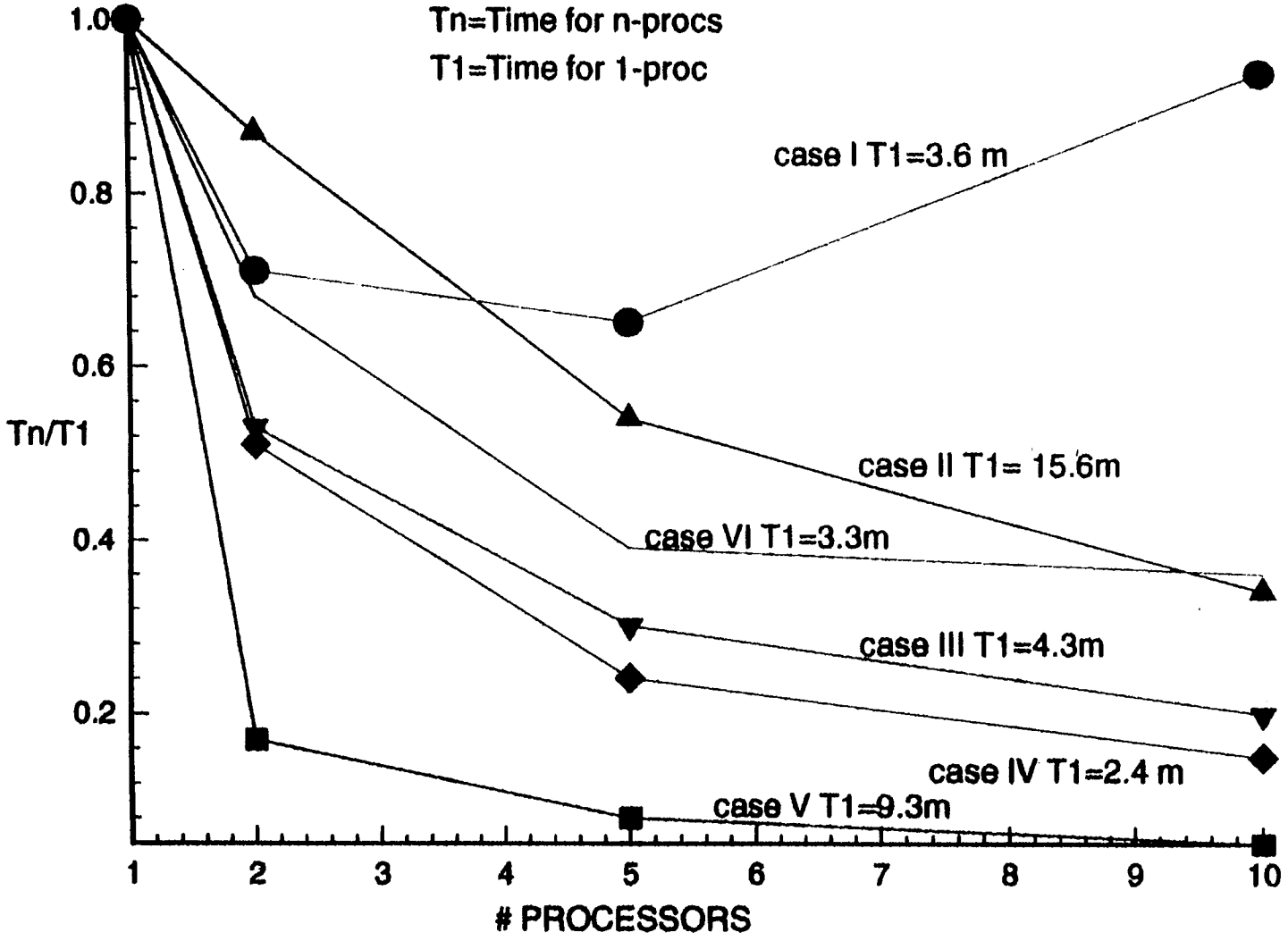


Figure 12. KSR1 test results using tiling strategy.

(2) For cases with larger memory than 30 Mb, the system was hung up due to data-swapping problems. For several cases listed later, a single processor run could not be performed due to this problem. The problem was more severe for double-precision computations. For this reason, a single-precision version of the model code was used after testing one case in double precision.

(3) For cases that had ample memory on one processor for the full problem, it was relevant to study the influence of decomposed memory vs. full memory in order to understand the degree of slowdown in performance. The data presented next would show that this depends on the array structure in the code.

(4) The model test problem does not account for communications between processors, since the model problem was fully decomposed and could be run independently on each processor. This is something that needs to be dealt with for the full CFD code later.

(5) The procedure was to run the code on 1–10 processors by decomposing the problem and its memory for all code structures that were studied for KSR1. The code tested was identical to that tested on KSR1.

8.1 Case I.

Array structure - y(3,kmax*jmax)
Results: 1 Proc time = 3:30 (m:s)
Memory - 30 Mb for 1 Proc

Procs	Time	Comment
1	3:30	For 10 procs using same storage as of 5 procs time is 0:39.
2	1:26	
5	0:42	
10	0:21	

8.2 Case II.

Array structure - y1(j,k), y2(j,k), y3(j,k), Mismatched loop index
Result: 1 Proc Time = 3:16

Procs	Time	Comment
1	3:16	Second set refers to memory kept at 1 proc level.
2	1:08, 2:05	
5	0:36, 0:55	
10	0:22, 0:47	

8.3 Case III.

Array structure - y1(k,j), y2(k,j), y3(k,j), Matched loop index
Result: 1 Proc Time = 2:03

Procs	Time	Comment
1	2:48	Second set refers to memory kept at 1 proc level.
2	1:09, 1:09	
5	0:39, 0:37	
10	0:21, 0:22	

8.4 Case IV.

Array Structure - y1(k*j), y2(k*j), y3(k*j), Matched loop index
Results: 1 Proc Time = 2:32

Procs	Time	Comment
1	2:32	—
2	0:59	
5	0:30	
10	0:19	

8.5 Case V.

Array structure - y1(k*j), y2(k*j), y3(k*j), Mismatched loop index
Results: 1 Proc Time = Hung up

Procs	Time	Comment
1	—	1 Proc was terminated at 12 minutes.
2	1:02	
5	0:33	
10	0:21	

8.6 Case VI.

Array Structure - y(3*kmax*jmax)
Results: 1 Proc Time = 2:34

Procs	Time	Comment
1	2:34, —	Second set is for double precision computations, 1 proc hungup.
2	1:0, 2:47	
5	0:31, 0:48	
10	0:20, 0:31	

8.7 Conclusions. The overall results shown above are pictorially depicted in Figure 13. For PARAGON, it appears that there is no clear advantage of one structure over the other. All arrays, whether linear or misaligned indices, seem to give very similar results as long as one ensures that the work assigned to a processor is within its local memory bound. This would imply that for the 2-D/3-D code, one may not have to lay out the data structure in any specific manner. Mismatched loop index makes a difference only when the work is at the limit of the memory bound for a given processor. Previous results show that, in this limit, the mismatched index style could be hung up while matched loop index style would complete its task.

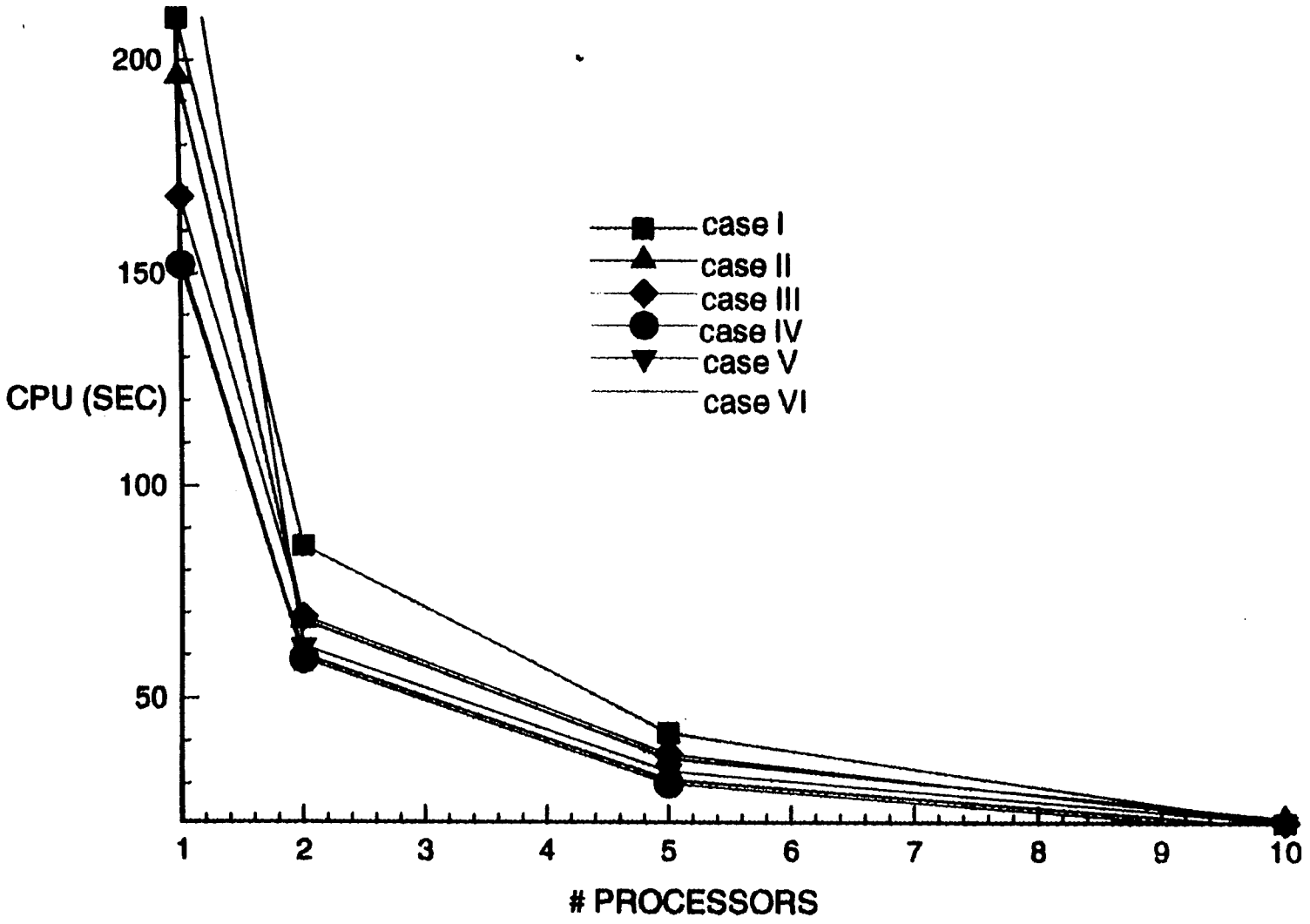


Figure 13. PARAGON test results for message passing.

9. CM-5 PERFORMANCE STUDY

The CM-5 at the Army High Performance Computing Center (AHPCC), managed by the University of Minnesota, does not allow individual processors to be used by the user. Processors in chunks of 32, 64, etc., however, can be requested. This implies that the model test problem size needs to be increased in order to get a valid comparison with other machines. However, since absolute comparisons are not intended here, it suffices to increase the problem size enough to get valid scaling. For all test cases reported here, the problem size was increased to jmax=40192, kmax=500.

Programming styles on CM-5 can consist of data parallel, message passing, and a combination of both. All programming styles were exercised on the model test and are described next.

9.1 Message Passing. The approach is similar to the PARAGON studies involving domain decomposition.

9.1.1 Case I.

Array structure - y(3,kmax*jmax)

Procs	Time (sec)	Comment
32 64 256	22.3, 33.7 10.6, 16.68 3.81, 5.44	Second set refers to double precision time.

9.1.2 Case II.

Array structure - y1(j,k), y2(j,k), y3(j,k), Mismatched loop index

Procs	Time (sec)	Comment
32 64 256	22.8, 21.1 10.1, 10.1 3.60, 3.9	Second set refers to memory kept at 32 proc.

9.1.3 Case III.

Array structure - y1(k,j), y2(k,j), y3(k,j), Matched loop index

Procs	Time (sec)	Comment
32 64 256	25.8 11.5 3.82	—

9.1.4 Case IV.

Array Structure - y1(k*j), y2(k*j), y3(k*j), Matched loop index

Procs	Time (sec)	Comment
32 64 256	18.6 8.7 3.4	Best timing of all cases.

9.1.5 Case V.

Array structure - y1(k*j), y2(k*j), y3(k*j), Mismatched loop index

Procs	Time (sec)	Comment
32 64 256	22.1 10.2 3.7	Mismatch makes a difference.

9.1.6 Case VI.

Array Structure - y(3*kmax*jmax)

Procs	Time (sec)	Comment
32	23.6	—
64	11.2	
256	3.9	

9.1.7 Conclusions. A critical look at the previous data and the pictorial representation in Figure 14 shows that CM-5 message passing programming does not show a great deal of differences for various array structures. The linear array structure with matched loop index seems to be doing somewhat better than other forms. Mismatching the loop index clearly shows a degradation in performance for both linear as well as 2-D arrays.

9.2 Data Parallel. The basic code, noted previously, was used via CM FORTRAN translator, CMAX, and the results are described next.

For all cases other than transparent 2-D arrays, CMAX failed to correctly translate the code into a data parallel style. The resulting .fcm file had to be manually modified for all other cases. These modifications yielded correct results, but long run times. In the process, a compiler error was discovered and reported. The results obtained after these modifications were unsatisfactory from a performance standpoint for cases that used linear-type arrays. The table in section 9.2.1 describes the result for Cases II and III.

9.2.1 Case II

Array Structure - y1(j, k), y2(j, k), y3(j, k)
CMAX translator worked well with this array structure.

Procs	Time (sec)	Comment
32	0.60	Run times are
64	0.30	excellent.
256	0.09	

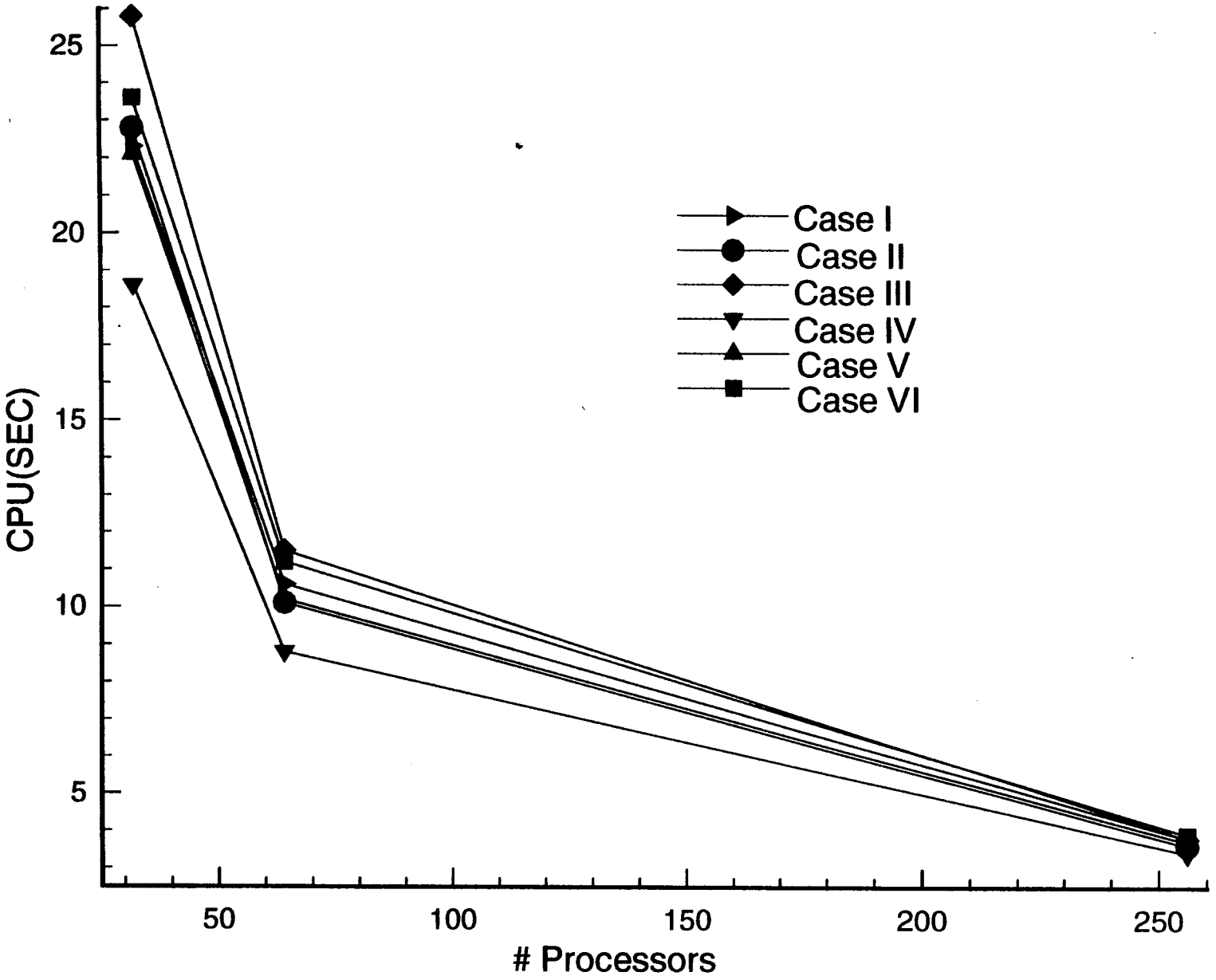


Figure 14. CM-5 test results for message passing.

9.2.2 Case III.

Array Structure - y1(k, j), y2(k, j), y3(k, j)
Same results as Case II.

9.2.3 Conclusions. The previous results show that CMAX can handle the 2-D arrays properly and the run times are excellent. For codes based on linear arrays, CMAX failed to translate the code correctly, yielding erroneous results. With manual intervention, a correct data parallel file was created for these cases but the run times were very high (of the order of several minutes). This implies that the codes based on linear arrays must be appropriately translated to 2-D/3-D arrays to take advantage of the existing CM-5 architecture.

9.3 Message Passing With Data Parallel. At a node level, data parallel programming can be used to combine the message passing programming to achieve greater performance. This, however, can not be done for all code structures since linear type data structure is not suitable for data parallel programming. A reasonable strategy is to use node level data parallel with inter-node massage passing. This will use the vector units at node levels which are never used in message passing programming. Only Cases II/III are relevant, and the results obtained are shown in section 9.3.

9.3.1 Case II/III.

Array structure- y1(k,j),y2(k,j),y3(k,j), Matched loop index

Procs	Time (sec)	Comment
32	1.8	Time for 256 procs is higher.
64	1.68	
256	4.87	

9.3.2 Conclusions. Data parallel message passing reduces the run time significantly as compared to the message passing. However, previous results also show that data parallel run times are the best of all programming styles. It would, then, suffice to conclude that 2-D/3-D type code structures should exploit the data parallel style to achieve optimum performance. Linear array type codes are likely to be easier to port if message passing style is used. Overall recommendation, however, is to first translate the code to 2-D/3-D array formats before attempting porting.

10. SUMMARY

This technical effort was initiated to evaluate various parallel computing platforms for large application codes. Specifically, the objectives were to develop an understanding of the enhancement in computing performance in relation to the code data structures. To this end, an emulated model code that has nearly all the features of the large code was tested on several massively parallel platforms such as KSR1, CM-5, and Intel PARAGON. This testing has provided several interesting conclusions that are enumerated next.

(1) The Intel PARAGON and the CM-5 model tests yielded the most consistent performance behavior on all code structures. For message passing, good run times (with correct results) were obtained for linear as well as 2-D arrays. For CM-5 data parallel programming, the array structures had to be limited to 2-D/3-D arrays. For these cases, all the performance data could be displayed on a graph with a narrow band width for various code structures implying robustness of the machine relative to the code structure.

(2) KSR1, although giving correct results on almost all code structures, suffered from performance inconsistencies which yielded widely varying run times for various code structures. Its performance curves could not be plotted on the same graph without normalizing the run times with one processor run time. This implies the sensitivity of the machine performance relative to the code structure. Use of the translation software, KAP, is strongly discouraged due to its inherent bugs.

(3) CM-5 performance based on data parallel programming seems to strongly favor the transparent 3-D/2-D arrays for data structures. Linear arrays of several variables yielded correct results with moderate run times; however, linear arrays where all variables are rolled into one seem to be lost in the system with very high run times. Use of CMAX translator should be limited to transparent data parallel programming styles.

(4) Of all systems, CM-5 seems to offer the wider selectivity of programming styles. Based on these studies, message passing with node level data parallel using transparent 2-D/3-D arrays appears to be the best choice for CFD applications.

(5) For Cray systems, C-90 and Y-MP, full 2-D/3-D CFD application codes were exercised with various code structures. With vectorization and multitasking, an estimated factor of 10 enhancement in code run time from the original code was obtained. Further code structure studies also show that 2-D/3-D array style of coding appears to give better performance on Cray systems.

(6) Overall conclusions based on these studies suggest that 2-D/3-D arrays are the most desirable choices for these classes of machines. To ensure portability to various machines, as well as rapid turn around on various platforms, use of linear arrays must be avoided.

(7) This modest effort is only a first step toward computing on massively parallel systems. Based on the lessons learned and conclusions derived here, further effort is underway to modify the current 3-D CFD code to evaluate its performance on KSR1, CM-5, and Intel PARAGON.

INTENTIONALLY LEFT BLANK.

11. REFERENCES

Kopper, F. C., R. Milano, R. L. Davis, R. P. Dring, and R. C. Stoeffler. "Energy Efficient Engine Component Development and Integration Program." NASA CR-165571, November 1981.

Srivastava, B. N. "Navier-Stokes Solutions for Highly Loaded Turbine Cascades." AIAA Paper No. 87-2151, AIAA/ASME 23rd Joint Propulsion Conference, San Diego, CA, June 1987.

Srivastava, B. N., and R. Bozzola. "Efficient and Accurate Numerical Solution of Euler and Navier-Stokes Equations for Turbomachinery Applications." AIAA 20th Joint Propulsion Conference, Cincinnati, OH, 11–13 June 1984.

Srivastava, B. N., and R. Bozzola. "Euler Solutions for Highly Loaded Turbine Cascades." Journal of Propulsion and Power, vol. 3, no. 1, p. 39, January–February 1987.

Srivastava, B. N., F. Maia, and J. P. Moran. "Computation of Viscous/Inviscid Flow-field in Pulsed Lasers." AIAA Journal, vol. 26, no. 10, pp. 1254–1262, October 1988.

Srivastava, B. N., F. Maia, J. Her, and J. P. Moran. "High Resolution Computation of Unsteady Flows." AIAA Journal, vol. 30, no. 3, pp. 756–764, March 1992.

INTENTIONALLY LEFT BLANK.

NO. OF COPIES	ORGANIZATION
2	ADMINISTRATOR DEFENSE TECHNICAL INFO CENTER ATTN: DTIC-DDA CAMERON STATION ALEXANDRIA VA 22304-6145
1	COMMANDER US ARMY MATERIEL COMMAND ATTN: AMCAM 5001 EISENHOWER AVE ALEXANDRIA VA 22333-0001
1	DIRECTOR US ARMY RESEARCH LABORATORY ATTN: AMSRL-OP-SD-TA/ RECORDS MANAGEMENT 2800 POWDER MILL RD ADELPHI MD 20783-1145
3	DIRECTOR US ARMY RESEARCH LABORATORY ATTN: A TECHNICAL LIBRARY 2800 POWDER MILL RD ADELPHI MD 20783-1145
1	DIRECTOR US ARMY RESEARCH LABORATORY ATTN: AMSRL-OP-SD-TP/ TECH PUBLISHING BRANCH 2800 POWDER MILL RD ADELPHI MD 20783-1145
2	COMMANDER US ARMY ARDEC ATTN: SMCAR-TDC PICATINNY ARSENAL NJ 07806-5000
1	DIRECTOR BENET LABORATORIES ATTN: SMCAR-CCB-TL WATERVLIET NY 12189-4050
1	DIRECTOR US ARMY ADVANCED SYSTEMS RESEARCH AND ANALYSIS OFFICE ATTN: AMSAT-R-NR/MS 219-1 AMES RESEARCH CENTER MOFFETT FIELD CA 94035-1000

NO. OF COPIES	ORGANIZATION
1	COMMANDER US ARMY MISSILE COMMAND ATTN: AMSMI-RD-CS-R (DOC) REDSTONE ARSENAL AL 35898-5010
1	COMMANDER US ARMY TANK-AUTOMOTIVE COMMAND ATTN: AMSTA-JSK (ARMOR ENG BR) WARREN MI 48397-5000
1	DIRECTOR US ARMY TRADOC ANALYSIS COMMAND ATTN: ATRC-WSR WSMR NM 88002-5502
1	COMMANDANT US ARMY INFANTRY SCHOOL ATTN: ATSH-WCB-O FORT BENNING GA 31905-5000
	ABERDEEN PROVING GROUND
2	DIR, USAMSAA ATTN: AMXSY-D AMXSY-MP/H COHEN
1	CDR, USATECOM ATTN: AMSTE-TC
1	DIR, USAERDEC ATTN: SCBRD-RT
1	CDR, USACBDCOM ATTN: AMSCB-CII
1	DIR, USARL ATTN: AMSRL-SL-I
5	DIR, USARL ATTN: AMSRL-OP-AP-L

NO. OF
COPIES ORGANIZATION

ABERDEEN PROVING GROUND

10 DIR USARL
ATTN AMSRL CI
WILLIAM H MERMAGEN SR
AMSRL CI C WALTER B STUREK
AMSRL CI CA
BARBARA D BROOME
B N SRIVASTAVA (5 CP)
DANIEL PRESSEL
AMSRL-WT DIXIE HISLEY

USER EVALUATION SHEET/CHANGE OF ADDRESS

This Laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers to the items/questions below will aid us in our efforts.

1. ARL Report Number ARL-TR-633 Date of Report November 1994

2. Date Report Received _____

3. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.) _____

4. Specifically, how is the report being used? (Information source, design data, procedure, source of ideas, etc.) _____

5. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate. _____

6. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.) _____

CURRENT ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

7. If indicating a Change of Address or Address Correction, please provide the Current or Correct address above and the Old or Incorrect address below.

OLD ADDRESS

_____
Organization

_____
Name

_____
Street or P.O. Box No.

_____
City, State, Zip Code

(Remove this sheet, fold as indicated, tape closed, and mail.)

(DO NOT STAPLE)

DEPARTMENT OF THE ARMY

OFFICIAL BUSINESS

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO 0001, APG, MD

Postage will be paid by addressee

Director
U.S. Army Research Laboratory
ATTN: AMSRL-OP-AP-L
Aberdeen Proving Ground, MD 21005-5066